Exercice 1

1. Montrer que si $c_n = \mathcal{O}(n^\alpha)$, alors $\sum_{k=1}^{n} c_k = \mathcal{O}(n^{\alpha+1})$.
2. Construire deux suites d'entiers (u_n) et (v_n), strictement croissantes, telles qu'on n'ait ni $u_n = \mathcal{O}(v_n)$, ni $v_n = \mathcal{O}(u_n)$.
3. Deux suites vérifient $u_n = \mathcal{O}(v_n)$. Est-il vrai que $e^{u_n} = \mathcal{O}(e^{v_n})$?

Exercice 2

1. Écrire une fonction récursive qui calcule la somme des n premiers entiers.
2. Déterminer sa complexité.

Exercice 3

On considère les fonctions :

```
let rec myst1 a l = match l with
   | [] -> [a]
   | x :: q -> x :: (myst1 a q);;

let rec myst2 l = match l with
   | [] -> []
   | x :: q -> myst1 x (myst2 q);;
```

1. Déterminer le nombre d'utilisation de l'opérateur `::` de la fonction `myst2`.
2. Déterminer le rôle des fonctions `myst1` et `myst2`.
3. Écrire une fonction ayant le même rôle que `myst2`, mais de complexité linéaire.

Exercice 4

On considère $(f_n)_{n\in\mathbb{N}}$ la suite de Fibonacci.

1. Montrer que pour $p, q \in \mathbb{N} \times \mathbb{N}^*$, $f_{p+q} = f_{p+1}f_q + f_p f_{q-1}$.
2. En déduire une fonction `fibo` permettant de calculer efficacement le terme f_n. On calculera sa complexité et on montrera qu'elle est meilleure que linéaire.
   Indication : selon la parité de n, on calculera un terme de la forme (f_{2k}, f_{2k-1}) ou (f_{2k+1}, f_{2k})

Exercice 5

1. Écrire une version récursive de l'algorithme d'Euclide.
2. En utilisant la suite de Fibonacci, déterminer sa complexité dans le pire des cas.

Exercice 6

1. Écrire une fonction récursive qui détermine si l'écriture en base 3 d'un nombre n ne contient que des 0 et des 1.
2. Déterminer la complexité de cette fonction.

Exercice 7

On propose l'algorithme suivant pour trier par ordre croissant un tableau :

```
let swap t i j =
   let x = t.(i) in
   t.(i) <- t.(j); t.(j) <- x;;

let tri t =
   let rec tri_aux i j = match j - i with
      | 0 -> ()
      | 1 -> if t.(i) > t.(j) then swap t i j
      | _ -> let k = (j + 1 - i) / 3 in
             tri_aux i (j - k);
             tri_aux (i + k) j;
             tri_aux i (j - k) in
   tri_aux 0 (Array.length t - 1);;
```

1. Prouver la correction de cet algorithme (c'est-à-dire montrer qu'il trie correctement un tableau).
2. On pose C_n le nombre maximal de comparaisons d'éléments du tableau effectuées lors du tri. Déterminer une relation de récurrence permettant de calculer C_n.
3. Calculer à la main C_n pour $n \in [\![1; 10]\!]$.
4. Montrer qu'à partir d'un certain rang n_0 à préciser, on a $C_n \geq n^2$.
5. Montrer qu'à partir d'un certain rang n_1 à préciser, on a $C_n \leq n^3$.
6. Justifier qu'il n'existe aucun $\alpha \in \mathbb{R}$ tel que $C_n \sim n^\alpha$.
7. Montrer qu'il existe un $\beta \in \mathbb{R}$ tel que $\forall \gamma > \beta$, $C_n = o(n^\gamma)$ et $\forall \gamma < \beta$, $n^\gamma = o(C_n)$.
8. Montrer que $C_n = \Theta(n^\beta)$.

Exercice 8

Le module graphique de `caml` s'ouvre en écrivant la ligne suivante (attention, le # est nécessaire) :

```
#load "graphics.cma";;
open Graphics;;
```

Avant de commencer à dessiner, il est nécessaire d'ouvrir une fenêtre graphique :

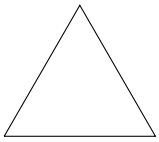
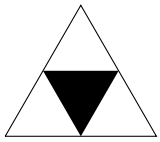
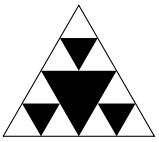
```
open_graph "";;
```

Dans la fenêtre graphique, il y a un point courant (initialement $(0, 0)$, c'est-à-dire dans le coin en bas à gauche), et on trace les lignes à partir de ce point. On trouve alors les commandes suivantes :

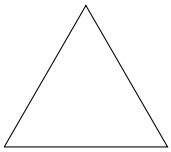
- `moveto : int -> int -> unit` : déplace le point courant aux coordonnées indiquées, sans rien tracer.
- `lineto : int -> int -> unit` : déplace le point courant aux coordonnées indiquées, en traçant le trait entre l'ancien point et le nouveau point.
- `rmoveto : int -> int -> unit` : déplace le point courant en le translatant d'un vecteur de coordonnées indiquées, sans rien tracer.
- `rlineto : int -> int -> unit` : déplace le point courant en le translatant d'un vecteur de coordonnées indiquées, en traçant le trait entre l'ancien point et le nouveau point.
- `fill_poly : (int * int) array -> unit` : trace et remplit le polygone formé par les points dont les coordonnées sont données par un tableau.

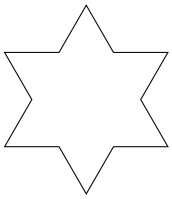
1. Tracer un carré de côté 50 pixels, dont le coin en bas à gauche est $(70, 70)$.
2. Écrire une fonction `sierpinsky : int -> unit` qui ouvre une fenêtre graphique et trace la fractale du triangle de Sierpiński à l'ordre donné en argument, selon le schéma suivant, qui représente

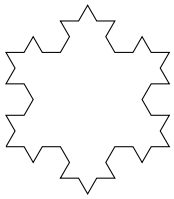
l'ordre 0, 1 et 2 :

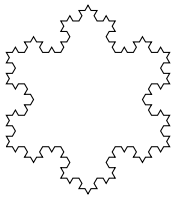




3. Déterminer la complexité temporelle de la fonction précédente.
4. Écrire une fonction `Koch : int -> unit` qui ouvre une fenêtre graphique et trace la fractale du flocon de Koch à l'ordre donné en argument, selon le schéma suivant :


Ordre 0


Ordre 1


Ordre 2


Ordre 3

5. Déterminer sa complexité.